Order Acceptance with Reinforcement Learning

M. Mainegra Hing, A. van Harten
& Peter Schuur

WP-66

BETA-publicatie : WP-66
ISSN : 1386-9213
NUGI : 684
Enschede : December 2001

Order Acceptance with Reinforcement Learning

M. Mainegra Hing* A. van Harten Peter Schuur

Dept. of Technology and Management,
University of Twente, The Netherlands

Abstract

Order Acceptance (OA) is one of the main functions in a business control framework. Basically, OA involves for each order a 0/1 (i.e., reject / accept) decision. Always accepting an order when capacity is available could unable the system to accept more convenient orders in the future. Another important aspect is the availability of information to the decision-maker. We use a stochastic modeling approach using Markov decision theory and learning methods from Artificial Intelligence techniques in order to deal with uncertainty and long-term decisions in OA. Reinforcement Learning (RL) is a quite new approach that already combines this idea of modeling and solution method. Here we report on RL-solutions for some OA models.

*corresponding author: P.O.box 217, 7500AE Enschede, The Netherlands, e-mail: M.MainegraHing@sms.utwente.nl

Contents

1 Introduction

Order Acceptance (OA) is one of the main functions in a business control framework. Basically, OA involves for each order a 0/1 (i.e., reject / accept) decision. Traditionally this problem is solved as follows: always accept an order if sufficient capacity is available. However, always accepting an order when capacity is available could unable the system to accept more convenient orders in the future. In Operations Research literature the idea of opportunity losses is recognized but not really worked out. Another important aspect is the availability of information to the decision maker. Generally in the literature information regarding negotiation with the customer such as an estimate of the work content of an order, a norm for the necessary processing time, the price and the due date are assumed to be known or estimated, and a model of the production process is also considered to be known beforehand. However it is difficult to obtain such information.

In this study we analyze the trade-off between long term opportunity costs and immediate yield in case of order acceptance under different degrees of uncertainty. In order to deal with uncertainty we use a stochastic modeling approach using Markov decision theory and learning methods from Artificial intelligence techniques. Reinforcement Learning is a quite new approach that already combines this idea of modeling and solution method. There are some aspects that make Reinforcement Learning (RL) appealing to our problem. The idea of learning without the necessity of complete model information and the possibility of learning even from delayed rewards allows us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. RL is quite a new field and successful applications are not always fully understood at a theoretical level. It means that convergence properties of the corresponding algorithms and procedures for tuning the parameters in the algorithms have to be explored. Hence, a lot of work still has to be done in order to understand how RL can best be applied to our field and to get insight into why some domains are considerably more tractable for RL than others.

1.1 General problem characteristics

Order acceptance is a typical decision making problem at the interface of customer relations management (CRM) and production management (PM). The rejection of an order may has further repercussions for the future customer relations. For an arriving order the implications of acceptance for production must be investigated, especially in terms of availability of production capacity. This leads to a certain projection for the delivery date for the order, which is communicated with the customer. Comparison with the customer due date demands may lead to agreement, perhaps after some iterative negotiations. The result of this due date setting process has again its effect on the customer relations. In case of acceptance, the next stage is the actual realization of the contractual order in a production process. The resulting earliness/tardiness of the actual delivery date is again a main element in terms of customer relation manage-

ment. Altogether, it is clear that CRM / PM coordination is the essence of order acceptance.

Order acceptance situations can be characterized with respect to
(i) customer flexibility in order definition,
(ii) predictability of order arrivals and order attribute parameters
(iii) effects of order loss
(iv) uncertainty in order characteristics for production,
(v) flexibility in capacity resources,
(vi) policy constraints.

In practice the situations encountered for order acceptance situations may vary with respect to each of the characteristics mentioned above. The market may be such that order definitions of quantity, price and delivery date are essentially nonnegotiable, specially if competition is fierce or they may be negotiable up to a larger degree. Predictability of orders may be stochastic or (partially) better due to customer consumption and/or replenishment patterns (say periodic), procurement contracts or logistical advantages (full truck or container loads). Order losses can lead to changes in arrival rates of orders and/or ordered volumes or they can be modelled with penalizations including estimates of future losses. Uncertainty in order characteristics is strongly dependent on the type of production environment. Compare a make to stock environment with an engineering to order environment. In the latter case, while discussing an order for acceptance part of the details for the realization phase may still be unknown, such as, e.g., product routings and capacity requirements. Flexibility in resources refers to capacity extension using overwork, hiring temporary flex labour staff or it addresses the issue of subcontracting parts of the order. Order acceptance policy constraints refers to a priori assumptions on the control structure. A policy constraint could be that under the circumstances that capacity is available and the due date can be met, an order is always accepted. Such policy corresponds to myopic management behaviour ([22]). Always accepting an order when capacity is available may bring the system in a bad situation for accepting more profitable orders in the future. It is an interesting facet of order acceptance policies to take such opportunity losses into account. How to find a good trade-off between long-term opportunity costs and immediate yield in case of order acceptance is a central problem in this paper. It complicates the order acceptance decision considerably.

Another important aspect of order acceptance decision making is the availability of information to the decision maker. It is evident that immediate information regarding negotiation with the customer such as an estimate of the work content of an order, a norm for the necessary processing time, the price and the due date is available. However, as for information on the state of the production facility several possibilities arise. They range from an estimate of the overall utilization rate to a detailed estimate of the evolution in time of the capacity profile for various workstations up to a planning horizon. Moreover the way in which uncertainty in this state information is dealt with may vary. Possibilities are that in some way slack is introduced or that at each time the state evolution is probabilistic. An analogous remark holds for the information on future or-

ders. For estimates of opportunity losses some information on arrival processes of orders is necessary. If not available yet there is the possibility to activate a learning process to overcome the incompleteness of the information. In practice such sort of information can be obtained by using adaptive statistical procedures or other learning techniques. Altogether this problem area constitutes a new and interesting field for several lines of research. In this paper we focus on the possible contributions of operations research and learning techniques for the development of decision support tools for order acceptance, from a management perspective, as described above.

1.2 Modeling order acceptance and learning

Our modeling approach is in line with the tradition of discrete time Markov decision processes in operations research. For an introductory survey we refer to Winston [25], section 21.5, where especially exercise 5 is interesting for our problem. As usual in Markov processes the system dynamics is described in terms of transition probabilities between a set of possible states. The effect of a decision d is expressed in terms of the transition probabilities from state i to j, i.e., it is modeled as $p(j|i,d)$. What one is looking for is an optimal decision making policy. Such a policy prescribes a decision $d(i)$ for each state i. Optimal refers to some cost or profit criterion which is usually taken as the total expected value of all future yields. A nice optimisation property for the so-called value function is available, Winston [25] and this gives rise to efficient algorithms to compute the optimal decision making policy, such as value iteration, policy iteration or reformulation as an LP problem.

This is a very flexible way of modeling order acceptance in the given context. The state definition combines information on the new arrivals of orders and their attributes with the characteristics of the capacity profile of the production, as mentioned in (i), (iv). The transition probabilities follow from the order arrival process in (ii), the production progress which includes new information due to better knowledge on the detailed design of orders, and the effect of decision making. In order to model the effect of acceptance of orders on production a priority rule for the adaptation of the capacity profile has to be assumed, say in agreement with a first come first serve or earliest due date principle. In a general setting also the effect of the decision making on capacity flexibility as in (v) can be shown in $p(j|i,d)$. Various sorts of assumptions on profits of accepted jobs, as well as costs of order rejection can be accommodated in the total expected value criterion.

Nevertheless the OR approach as sketched has some shortcomings. Firstly, though this sort of modeling is very flexible, it assumes a priori knowledge on many parameters and if this information, say on $p(j|i,d)$, is incomplete, the method has to be adapted. This can be done in a two phases procedure where in the first step the necessary information on parameters is gathered with a stochastic decision policy and next, once the information on parameters is sufficiently accurate, the optimization of the decision policy takes place. Secondly, the Markov decision approach suffers from what is usually referred to as the curse

of dimensionality. In a somewhat complex business environment the dimension of the state for order acceptance problems may easily be astronomically large. This asks for the introduction of approximations. Therefore it also becomes an issue that the learning technique is compatible with effective approximation strategies.

For both these reasons we explore in our research a new alternative for effective process control strategies in order acceptance problems by using intelligent computational techniques, particularly Reinforcement Learning (RL), also known as Neurodynamic Programming. These techniques have been proven to be successful in distilling information from large amounts of data. The distilled information can be used to learn about the opportunity costs or incomplete information and to optimize the process control under study. Reinforcement Learning is a rather new approach that can be interpreted as a conjunction between learning machine problems (automatic goal learning) and Markov decision models (decision making problems). RL focuses on an agent (a virtual decision maker) with a defined goal (optimization criterion) that through trial and error in interaction with its environment (in our case the CRM/PM interface) tries to learn optimal behaviour. This means decision making such that the criterion is optimized in the long run. Once the agent is properly trained it constitutes a decision support tool for the order acceptance management. There are some aspects that make RL appealing to our problem. The idea of learning without the necessity of complete model information and the possibility that the agent learns even from delayed rewards (when the effects of an action can be known only in the future) allow us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. Moreover, RL combined with the potentialities of neural networks has been claimed to overcome the curse of dimensionality as it appears in many complex problems of planning, optimal decision making, and intelligent control, also in our problem setting. It is worthwhile to investigate this claim. After all, RL is quite a new field and successful applications are not always fully understood at a theoretical level. It means that convergence properties of the corresponding algorithms and procedures for tuning the parameters in the algorithms have to be explored. Hence, a lot of work still has to be done in order to understand how RL can best be applied to our field and to get insight into why some domains are considerably more tractable for RL than others.

Summarizing, the new elements for order acceptance in this paper concern:

- introducing some new Markov decision models and analyzing optimal policies
- exploring how to deal with at the onset incomplete information on transitions
- how to use RL in this case
- evaluate RL, gain insight and find generalizations.

In order to do so we shall focus on a prototype form of the order acceptance problem for reasons of transparency. Mainly this means that we consider orders that can be classified according to a finite number of classes and the production capacity is modeled as just one resource. Generalizations to more complex order acceptance situations will be discussed only briefly, in less detail, also since that work is still in progress.

1.3 Discussion on some relevant literature

Relatively little attention has been paid to the order acceptance problem in the literature. Nevertheless, relevant literature can be found in the area of operations management concerning the CRM/PM interface, in the area of Operations Research (OR) and in the area of Reinforcement Learning (RL).

In the first area some studies have been done about the degree of information required to deal with the coordination mechanism between the order acceptance function and the scheduling function. Three policies were compared by Wester, Wijngaard and Zijm [22]. (1)The monolithic policy accepts orders based on a detailed schedule which is built upon an order arrival. (2)The hierarchical and (3) myopic policies take their decision based on the total workload of all accepted orders. Next the hierarchical policy makes a detailed schedule with the accepted orders, whereas their myopic policy uses some simpler dispatching rules. The experimental results showed that in situations with large setup times and tight due dates the monolithic approach performs better, probably due to a phenomenon of implicit selective acceptance. Moreover they found no big difference between the hierarchical and myopic policies. The hierarchical and integrated approaches were also compared by Ten Kate [5] in process flow industries. He explains why due to uncertainty and complexity of production, hierarchical structures are more widely used. The experimental results show that only for tight situations (short lead times, high utilization rate) the integrated approach outperforms the hierarchical approach although still in such situations the performance is often bad for both approaches. The problem of accepting orders together with capacity loading decisions is studied in multi-purpose batch process industries by Raaymakers [12]. Besides the traditional workload and scheduling policies she also considers a makespan estimation policy which uses some aggregate information about the current job mix. From the empirical results she obtained that for situations with high demand and product variety, in which detailed information can be difficult to obtain, the makespan estimation policy can be a good alternative due to its better performance compared to the workload policy. Altogether this line of work is oriented mainly towards due date performance, not so much towards costs and profits, certainly not opportunity losses.

In the operations research literature the idea of opportunity losses in order acceptance problems is recognized, but not worked out in as full generality as we propose. This problem arises naturally in the context of reservation systems for car rentals, room reservations in hotels or tank capacity rental. Usually such problems are discussed under interval scheduling, Pinedo et al [10] but

the problem is then deterministic. Opportunity costs are analyzed as part of relevant costs for order acceptance decisions by Wouters in [26]. Relevant costs for order decisions are considered to be those avoidable by rejecting an order. Guidelines on the necessary information from production planning and control function are given to calculate relevant costs and also to assess the reliability of such calculations under uncertainty. A single server system in continuous time in which opportunity costs play a role is studied by Nawijn [7]. A decision has to be made between starting a new service for an arriving order or rejecting an arriving order depending on its expected processing time. Orders that arrive while the server is busy, are lost. He proves that there exists an optimal control policy that maximizes the expected average number of customers per unit of time. Also in the field of OR, Garbe [4] made an interesting deterministic description of some heuristic online[1] order acceptance algorithms. Although some algorithms are given, the results are mainly theoretical and the theorems are concerned with a worse case analysis and for a restricted set of problems.

In this study we consider a more general problem setting regarding order attributes and service capacity and with different degrees of uncertainty as explained in the previous subsection. Problems get more complex when one considers uncertainty, which is closer to what happens in the real world. Generally in the literature a model of the production process is considered to be known beforehand. The problem is that most of the time such a model is hard to find. Sometimes even an approximate model of the process is not easily obtained due to incomplete information. We consider Reinforcement Learning as a remedy.

Reinforcement Learning has already been used in some logistic problems like job-shop scheduling [27], elevator control [3], resource allocation [15], inventory control. In all these problems the dynamic model was known beforehand, but the size of the problems made them intractable for traditional dynamic programming methods. The system behaviour is learned through simulations, and its performance is improved by means of iterative reinforcement. The study of order acceptance policies in a production environment using RL is new. Moreover, the objective of this introductory study is to determine the possibilities of neuro-dynamic methods as an effective approach for order acceptance problems with incomplete, dynamic and generally uncertain information. This aspect makes this study even more challenging, since it is not so explicitly explored and compared with more traditional methods in the literature, as far as we know.

1.4 Brief survey of the contents

In the next section we introduce some simple versions of our problem in terms of the Markov decision theory. First we explain some details of the models with complete information. The optimality equations are introduced and the structure of the optimal decision policy is discussed. Next, we discuss some

[1] An online algorithm has to decide whether or not to accept an arriving order immediately upon its start time without any further information about possible successors.

solution methods for the problem. Finally we describe some possible extensions. In Section 3 we specify the Reinforcement Learning techniques used. Several versions of so-called $Q(\lambda)$-learning methods are introduced. A key issue is the parameter setting for the learning procedures. In Section 4 we report on a series of experiments to show the effect of the influence of the various parameter settings on the speed of convergence of the methods. As a result, the most appropriate variant of the RL methods can be identified. Also some results with the two phase OR approach are presented. In the last section we give our conclusions, summarize the acquired new insight and formulate some open problems and ideas for further research.

2 Modeling the Order Acceptance Problem.

In terms of the characterization (i)-(vi) given in the introduction we assume the following: (i) order definitions are based on a finite number N of job types, where each type has a specific processing time and immediate reward, (ii) at most one arrival in each discrete time unit and the conditional probability of arrival is job type dependent, (iii) rejection of an order affects only the immediate reward of that order (iv) job processing requires a job type dependent deterministic service time from a single server (v) only one server is available, hence only one job can be processed at any moment in time (vi) the decision policy does not allow preemption of jobs and when the server is busy the only option for a new order is rejection. Note that in our prototype case orders have the smallest possible set of attributes, for example the concept of a due date does not play a role (conceptually it coincides with the processing time here), and batch arrivals of orders are excluded. Production capacity is modeled as a single server without a queueing facility.

Our problem can be represented as a Semi-Markov Decision problem (SMDP). Therefore some concepts and notations for SMD models are first introduced in Section 2.1. Then we describe the prototype case from the SMD perspective in Section 2.2. The optimality equations for the decision policy are introduced and the structure of the optimal decision policy is discussed. Methods from Markov Decision theory can easily cope with this problem in case the complete model is known, otherwise, in case of incomplete information on the parameters of the problem, new alternative methods (as those presented in Section 3) should be applied.

In the context of SMDP theory models of more realistic order acceptance problems can be studied. As mentioned before generalizations with respect to any of the characteristics (i)-(vi) in the prototype case are interesting from a point of view of applications in practice. In Section 2.3 we explore a model formulation, which takes into account (i) orders with a larger attribute set including due dates,(ii) batch arrivals, (iv) some uncertainty in job processing and(v) a more general capacity structure in production. In general this leads us to multi-server (parallel and/or in series) network problems with waiting lists (queueing) per server (or a common list for a set of servers) with batch arrivals

of orders. In this paper we restrict ourselves to the case where the production is modeled as one workstation consisting of a number of parallel servers and we suppose that order queuing, i.e. delay of acceptance decisions is still not allowed. Nevertheless this more general case is rich enough to demonstrate the complications that arise in finding an optimal decision policy. Work on more general problems is still in progress and we hope to report on it in the near future.

2.1 Semi-Markov Decision basis

Semi-Markov Decision models are models for sequential decision making in dynamic systems under uncertainty (Puterman [11]). In general a semi-Markov decision model is characterized by the following elements:

- The state space characterizing the system: S
- The action space defining the decisions that can be taken: A
- The decision epochs are the moments in time at which decisions have to be taken. In general the next decision epoch depends on the present state and the action; and decision epochs do not necessarily occur at equidistant points in time. Hence in the general case a next decision epoch lies $d(s,a)$ in the future, where s refers to the present state an a is the taken action under the current state. When the next decision epoch is independent of the current state and the action with the consequence that decision epochs are equidistant in time the process is called Markovian.
- The transition probabilities describe the dynamics of the system.

  $p(s,a,s\prime)$ is the probability that , when the action a was chosen in the actual state s, at the next decision epoch $d(s,a)$ time units ahead, the next state is $s\prime$.

- The immediate reward function $rew(s,a)$ is the reward received after having taken decision a in state s.
- The optimality criterion defines a performance measure of the system. The most common optimality criterions are:
  - Total expected reward for a finite planning horizon.
  - Total expected discounted reward over an infinite planning horizon.
  - Average expected reward per unit of time.

In this paper we consider the total expected discounted reward. A discount factor $\gamma < 1$ corresponds to the idea that rewards are less attractive in the far future. Given a policy $\pi : S \rightarrow A$, the state-value function V^{π} is defined for each

state s as the total expected discounted reward starting in state s and following policy π:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{T_k}r_k|s_0 = s\right\} \tag{1}$$
$$= rew(s,\pi(s)) + \gamma^{d(s,\pi(s))}\sum_{s'}p(s,\pi(s),s')V^{\pi}(s')$$

where r_k is the immediate reward at the decision epoch k and T_k is the elapsed time between $t_0 = 0$ and the $k-th$ future decision epoch. The objective is to find the optimal policy $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$.

From the theory of the discrete SMDP it follows that there exists an optimal stationary decision policy. This optimal policy can be found by any of the traditional methods from Probabilistic Dynamic Programming in the case of complete information about the model. For more on these methods see Winston [25] and Puterman [11]. Linear programming, value iteration and policy iteration methods are the most widely used, and they are basically based on the Bellman equation (1) where all the details about the model are assumed to be known. The value for the optimal policy $V^*(s)$ satisfies the following relation:

$$V^*(s) = \max_{a}\left[rew(s,a) + \gamma^{d(s,a)}\sum_{s'}p(s,a,s')V^*(s')\right]$$

So far we presented the classic SMDP setting. Traditional solution methods for this problem require full knowledge about the model parameters (i.e., transition probabilities, reward function, etc.). However, from previous work of the Artificial Intelligent community on learning optimal decision policies a slightly different concept, the action-value function Q^{π}(Watkins [21]) appeared to be more advantageous. This function is defined for each pair state-action (s,a) as the total expected discounted reward starting in state s , taking action a and thereafter following policy π:

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{T_k}r_k|s_0 = s,\ a_0 = a\right\} \tag{2}$$
$$= rew(s,a) + \gamma^{d(s,a)}\sum_{s'}p(s,a,s')V^{\pi}(s')$$

where $V^{\pi}(s) = \max_{a} Q^{\pi}(s,a)$.

In terms of this action-value function under an optimal policy π^* the optimality relation is given by

$$Q^*(s,a) = rew(s,a) + \gamma^{d(s,a)}\sum_{s'}p(s,a,s')\max_{a'}Q^*(s',a')$$

The importance of the action-value function $Q^*(s,a)$ is to remedy the insufficiency of the state-value function $V^*(s)$ to reconstruct the optimal policy

purely from its values. The optimal policy follows directly from the optimal action-value function through:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3}$$

Note that in this relation there is no reference to other information than $Q^*(s,a)$ itself. This relation can easily be reformulated in terms of $V^*(s)$ as

$$\pi^*(s) = \arg\max_a \{rew(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') V^*(s')\}$$

but to use this formula we need a lot of model information, since all model parameters enter in the formula besides the value function. In problems with incomplete information this is a disadvantage for formulating transparent learning algorithms. Therefore, Section 3 is based on learning the function $Q^*(s,a)$. Once a good approximation is found the corresponding policy follows directly from the relation given above.

2.2 The Prototype problem as a SMDP

In this Section we formulate a simple prototype order acceptance problem. as a SMDP. In this order acceptance problem production capacity is considered as a single server that can only process one job at a time. Orders arrive in a single arrival process from a set of N order types. Order type j is characterized by a small set of attributes:

- t_j: processing time,
- r_j: reward for acceptance,
- p_j: arrival probability

Arrivals are checked every fixed period of time, and a decision should be taken immediately at that moment if the server is idle: accept or reject the arrived order. If the server is busy, the arrived order is lost. Without loss of generality, it can be assumed that one job is arriving in every time unit by introducing a dummy type of job, with processing time equal to 1 and 0 reward. The goal is to take decisions that maximize the total expected reward in the long run. This prototype problem can easily be modeled as a discrete time Semi Markov Decision Problem (SMDP) as sketched before. State s identifies the arriving order type and there are two possible actions, reject or accept ($a = 0,\ a = 1$) since decision moments occur by definition only when the server is idle. The state transition is described in the table below:

actual state	s	
action (a)	0	1
immediate reward $rew(s,a)$	0	r_s
time until next decision $d(s,a)$	1	t_s
next state	s'	
transition probability $p(s,a,s')$	$p_{s'}$	

The objective is to find a policy π:

$$\pi(s) = \begin{cases} 1 & \text{accept the arriving job } s \\ 0 & \text{reject the arriving job } s \end{cases}$$

which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward. The Bellman equation for the value function V^π of policy π is given by

$$V^\pi(s) = rew(s,\pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} p_{s'} V^\pi(s')$$

with γ the discount factor. The corresponding optimal value function $V^*(s)$ satisfies:

$$V^*(s) = \max \left[\gamma \sum_{s'} p_{s'} V^*(s'), \quad r_s + \gamma^{t_s} \sum_{s'} p_{s'} V^*(s') \right]$$

The optimal policy in this case has a very special form: an order of type s should be accepted if and only if $r_s \geq (\gamma - \gamma^{t_s})\beta$, where $\beta = \sum_s p_s V^*(s)$ is defined by the problem instance. Hence this policy has a nice simple structure.

The action-value function $Q^\pi(s,a)$ is defined as:

$$Q^\pi(s,a) = rew(s,a) + \gamma^{d(s,a)} \sum_{s'} p_{s'} V^\pi(s')$$

and the optimal Q^* satisfies:

$$Q^*(s,0) = \gamma\beta$$
$$Q^*(s,1) = r_s + \gamma^{t_s}\beta$$

The policy iteration algorithm finds the optimal policy π^* for this problem in at most N steps. Since we want to look at this problem from the incomplete information perspective, in general these traditional methods do not fit into our view. A straightforward way to deal with incomplete information is a two phase method. In the first phase the unknown parameters of the model are estimated. In the second phase, the complete problem with the estimated parameters is solved. In this problem estimation of parameters boils down mainly

to estimation of the transition probabilities. Since the essence of the transition probabilities are the arrival intensities, this can simply be done using the historical arrival fraction of jobs of type j as an estimator for p_j. Other methods that combine learning and solving without separation in two phases are those which attempt to learn action-value functions (Q-learning). We present these ideas in Section 3 and we show some computational results in Section 4.

2.3 A more general problem with due dates and multi-server capacity

In this Section we consider an extension of the previous prototype model for order acceptance with multiple arrivals, with due dates and with multi-server capacity planning.

Concerning arrivals of orders we now assume:

- N types of jobs, each with independent Poisson arrival processes. Job type i is characterized by the following attributes
  - arrival rate λ_i,
  - due-date t_i,
  - requested capacity w_i,
  - reward r_i.

As for the production we now assume:

- A capacity profile with:
  - a planning horizon of $H = \max_i t_i$ stages, where each stage spans one time unit,
  - a maximum capacity per stage $C_{\max}$,

    for each stage j there is a planned capacity utilization $c_j, (0 \leq c_j \leq C_{\max})$ and the capacity profile is defined by the vector $(c_1, ..., c_H)$. Figure 1 shows an example.

We assume without loss of generality that the beginning of each time unit is a decision epoch. The present decision epoch will usually be referred to as $t = 0$. For each given decision epoch we assume that arriving orders occurring after the previous decision epoch are accumulated into a list. Each order on the list has to be judged with an acceptance or rejection decision, as a result delaying a decision for a job on the list is not an option. In this case the number of possible decisions may be considerably larger than in the prototype problem. Now, a decision has to be taken about which subset of the jobs requesting service to accept. In principle each subset of jobs should be analyzed to see whether it fits into the available capacity in such a way that the due dates are not violated. As

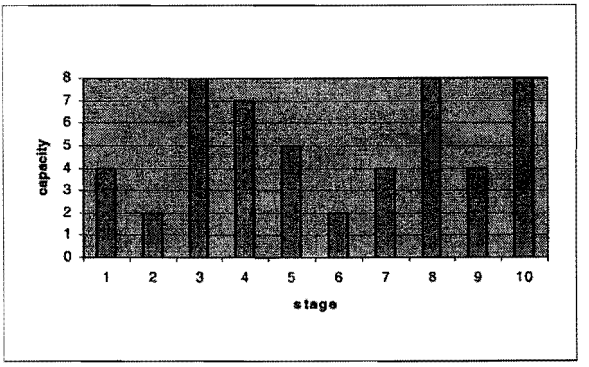

Figure 1: Capacity utilization in a planning horizon of $H = 10$ stages and $Cmax = 8$.

in the prototype problem, a subset of jobs can also be rejected even when there is sufficient available capacity in order to save capacity for more profitable future jobs. Note that in this view the amount of possible decisions is non-polynomial in the length of the order list, the number of job types and the available capacity, in general.

In our approach we want to reduce the number of possible decisions to a polynomial size in the number of job types. This is be done by imposing some restrictions on the structure of the decision rule. Instead of focussing on all possible subsets of jobs at once as possible decisions, we impose that the decision is created sequentially, as a sequence of sub-decisions while we take a time off. Each sub-decision a consists of the selection of one of the jobs from the remaining list of jobs requesting service ($a \in [1..N]$) or to reject all of them ($a = 0$). By definition we put $a = 0$ also if the remaining list is empty. If the available capacity is not enough for a job, then the selection of that job is not an option. When one of the jobs is selected ($a > 0$) then capacity is allocated to this job and the list of remaining jobs is updated. If $a > 0$ then we go to the next sub-decision for the same decision epoch. If $a = 0$ we go to the next decision epoch (the time move on again) with a new list of newly arriving orders. From now on we name the sub-decisions as decisions.

The dynamics of the system is strongly dependent on how the capacity planning is done. We shall use a fixed prescription for the capacity planning, which corresponds to the highest degree of flexibility. Therefore optimizing the capacity planning is not an issue. The non-allocated capacity is available for executing new orders. The requested capacity for a job can be freely allocated within the available capacity before its due date. For example if in a situation like in Figure 1 a new job is accepted with a due date 8 and a capacity requirement of 5 units, it is allowed to plan the execution of that job "just in time " (JIT) providing 4 capacity units at time 7 since stage 8 is already fulfilled, and 1 capacity unit at time 6 as in Figure 2. Essential is how the capacity profile is updated given the

decision a (I) of accepting a job of type a ($a > 0$), or (II) rejecting the complete job list ($a = 0$). Both for (I) and (II) capacity allocation may be done following different planning rules. (I) and (II) are different from a planning perspective. In case (I) it is logical to choose for a JIT principle, since allocating capacity as close to the due date as possible provides the best conditions for accepting a next job from the list. This principle boils down to loading within the available capacity backwards in time starting from the due date. Case (II) is completely different. If there is still available capacity in the first stage ($t = 1$), then it is lost unless we adapt the allocation. We may create the best conditions for accepting jobs from the new list at the next decision epoch, if we fill the still available capacity at $t = 1$ according to a "least shift back " (LSB) principle. This boils down to looking for already allocated capacity forward in time starting at $t = 2$, which is replanned to $t = 1$ until the still available capacity at $t = 1$ is filled as much as possible.

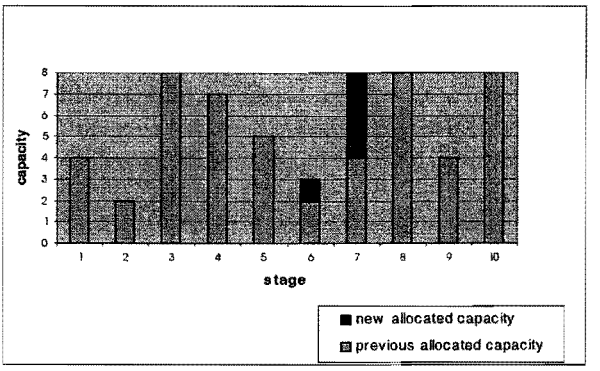

Figure 2: Allocate capacity=5, due date=8

We also introduce an element of uncertainty in the dynamics related to perturbed processing of the jobs. Therefore we introduce a stochastic perturbation (p) in the evolution of the capacity profile. For simplicity we consider a random perturbation term $p \in \{-1, 0, 1\}$ which occurs only when the time move on to a new stage ($a = 0$) and affects the capacity profile only for what is currently in $stage = 1$. As described in (II) this perturbation propagates to the next decision epoch. With $p = 0$ we refer to the unperturbed situation, where work is executed as planned. In case of perturbation the work planned into stage 1 requires more service capacity than anticipated ($p = 1$), or the work requires less service capacity than anticipated ($p = -1$). A perturbation of $p = -1$ gives one extra capacity unit available, if the already allocated capacity at that time is positive and a perturbation of $p = 1$ takes one capacity unit away from the available capacity, if it is positive. If $p = 1$ and no capacity is available we

introduce a penalty $pen(c,p)$ for using non-regular capacity.

$$pen(c,p) = \begin{cases} -\eta, & c_1 + p > C_{\max} \\ 0, & \text{otherwise} \end{cases}$$

with η>0.

We can now formally describe this generalized problem as a Semi Markov decision problem. The state at each decision moment is defined by $s = (k, c)$, where

$k = (k_1, ..., k_N)$ is the job list and k_i represents the amount of jobs of type i requesting service,

$c = (c_1, ..., c_H)$ is the capacity profile and c_j is the total capacity already allocated in stage j.

The presence of a list with k_i jobs of type i at a certain decision epoch has the following joint probability distribution

$$prob\{new\_arrivals = (k_1, ..., k_N)\} = \frac{\lambda^k}{k!} \exp(-\Lambda)$$

where we use the shorthand notation $\lambda^k = \Pi\lambda_i^{k_i}$, $k! = \Pi k_i!$ and $\Lambda = \Sigma\lambda_i$.

The possible actions at each decision moment depend on the current state s. Action $a \in [1..N]$ is possible if order type a is present in the order list and capacity is available for that job type (i.e., $a \in A(s)|\ A(s) = \{a \in [1..N]\ |k_a \neq 0, JIT(c,a)$ is possible$\}$. The procedure $JIT(c,a)$ computes the capacity planning as discussed in (I). First it assesses the possibility to allocate job a in the capacity profile c. If it is possible the procedure returns the new capacity profile otherwise a *not possible* answer is given. This procedure is sketched in Figure 3 [2]. After the job a is selected the new job list is given by $k - e_a$ where e_a is a unit vector with 1 in position a. Notice that once a job is selected and allocated, the transition to the next state is deterministically determined by the JIT procedure. In case of rejection of all remaining jobs ($a = 0$) we are in situation (II), where a new list of arriving jobs is considered in the next decision epoch. The new arrivals vector k' is determined by the jobs arrival process with statistics as described before. Given the current capacity profile c and the perturbation term p function $LSBp(c,p)$ updates the capacity profile for the next moment as shown in Figure 4. The state transition is described in the table below:

s	(k, c)	
a	$a \in A(s)$	0
rew(s,a)	r_a	pen(c,p)
d(s,a)	0	1
s'	(k-e_a, JIT(c,a))	(k', LSBp(c,p))
p(s,a,s')	1	Pr{k'}Pr{LSBp(c,p)}

Procedure JIT (c,a)

$i=t_a$,

requested_capacity=w_a

$c'(t_a+1:H)= c(t_a+1:H)$

while $i \geq 1$ and requested_capacity$\neq 0$

θ=min(requested_capacity, C_{max}-c_i),

$c'_i=c_i+\theta$,

requested_capacity= requested_capacity- θ

i=i-1,

if requested_capacity $\neq 0$

return('not possible')

else

return(c')

Figure 3: Just in Time allocation procedure for job type a in a capacity profile c.

The objective is to find a policy π:

$$\pi(s) = \begin{cases} a & a \in \overline{1..N} \text{ selecting job type } a \\ 0 & \text{reject the remaining jobs} \end{cases}$$

which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward, the Bellman equation for the value function V^π is given by:

$$V^\pi(s) = rew(s, \pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} p(s, \pi(s), s') V^\pi(s') \tag{4}$$

with γ the discount factor.

The optimal Q^* in this case satisfies:

$$Q^*(s,a) = rew(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') \max_{a'} Q^*(s',a') \tag{5}$$

and as before, the optimal policy $\pi^*(s)$ can be determined by:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

[2]Here we used the notation $x(i:H)$ to denote the sub-array of $x(1:H)$ starting with the $i-th$ element.

Procedure LSBp(c,p)

extra_capacity=max(0,C_{max}-max(0,c_1+p))

c_{H+1}=0,

i=2,

while i≤H+1 and extra_capacity>0

θ=min(ci, extra_capacity)

c'_{i-1}=c_i- θ,

extra_capacity=extra_capacity- θ,

i=i+1,

if extra_capacity=0

c'(i-1:H)= c(i:H +1)

return(c')

Figure 4: Least Shift Back plus capacity perturbation allocation procedure for capacity profile c and perturbation p.

In this problem even in the case that the parameters of the model are known (exactly or by some statistical estimation) solving the problem is still a difficult task due to what is usually referred to as the "curse of dimensionality". It is well known that finding an optimal policy requires an overwhelming computational effort if the dimension of the state space increases. Here the state space may be tremendously large $((Cmax+1)^H \prod_{i=1}^{N}(m_i+1)$ where m_i is the maximum number of arrivals of type i to be considered, more jobs over this amount are neglected). As a consequence $\pi^*(s)$ may be a very complex rule. A simple structure as found in the prototype problem may no longer be expected. What one should expect is, that in case of low utilization of the capacity an optimal decision rule will be inclined to accept more jobs which generate low profit per unit of time than in case of high utilization. Further one should expect that even in case of high utilization certain small rush jobs, that fit well into a gap in the capacity profile are still attractive for acceptance, even if they are not so profitable, simply because the gap is too small to accommodate other jobs. An interesting aspect of this research is to see upto what degree the learning approaches lead to an approximate decision rule representing such effects.

In the next Section we explore new alternative methods that work with a reduced state space and approximated Q-functions. Particularly learning methods like Reinforcement Learning that can cope with both issues (incomplete model information and approximation) are interesting. These methods do not assume an a priori model and they can construct good policies without necessity of learning model parameters. However hybrid approaches in which learning pa-

rameters of the model is an explicit part of the learning process, are also be tried. Such learning techniques are introduced in Section 3 and results are presented in Section 4.

3 Reinforcement learning: solving SMDP with incomplete information

In the previous sections we have argued that in order to support order acceptance decisions we need to explore new alternatives able to cope with long term opportunity costs and uncertain environments with incomplete information. In this section we deal with one of such possible alternatives: Reinforcement learning (RL). It is about the interaction between an active decision-making agent and its environment within which the agent seeks to achieve a goal despite uncertainty about its environment, which can be described as a Semi-Markov decision process.

An attractive feature of RL is that little knowledge is needed in order to apply it to a given task. That is why the scope of applications is quite broad. The areas that appear to be the most popular are intelligent control, robotics, game-playing, and combinatorial optimization. The famous self-learning backgammon playing program by Tesauro, TD-Gammon [18], is still one of the most successful application of RL.

Basically in RL methods, parameters of the problem can be learned or direct policies can be learned without necessity of learning the underlying model itself. Here we discuss one of these methods, Q-learning, directly applied to our order acceptance problem. First in subsection 3.1 we introduce some basic RL concepts and we address some interesting questions.

Although an accepted method for solving sequential decision making problems, Reinforcement Learning methods still presuppose some questions for one who wants to use this technique. Sutton [14] makes an analysis of some open questions in the RL theory. Which from the variants fix better to the problem at hand? How to tune the parameters involved in the learning mechanism in order to obtain an efficient algorithm? In subsections 3.2 and 3.3 we analyze these questions related to the two models introduced in the previous Section.

As a trial-and-error method, Reinforcement Learning can be limited for real world problems where making error have high cost, mainly if the learning lasts too long. Model based methods are a way of incorporating knowledge in a helpful way and can be used to speed up Reinforcement Learning. Some model based methods are introduced in subsection 3.4.

In the next Section we present experimental results for the topics discuss in here.

3.1 Reinforcement Learning: Basic concepts

Reinforcement Learning studies an intelligent system (agent) learning to achieve a goal through interaction with its environment. The idea of learning from inter-

action started in the early days of artificial intelligence but it was not thoroughly explored till the last 15 years and it has become one of the most active areas in machine learning. In previous machine intelligence paradigms like symbolic processing or supervised learning the intelligent system is supposed to receive some knowledge information beforehand, process such information and use it as his knowledge base. Unfortunately there are many situations where we do not have this a priory information. In the paradigm of Reinforcement Learning, the intelligent system is supposed to gather such information sequentially at the same time it is learning from the gathered information. The collection of information and the learning process occur simultaneously by the interaction between the intelligent system and the subject that is to be learned. The information the agent receives does not need to be in the form of complete rules (the well known IF_THEN rules) as in a symbolic processing paradigm or like perfect match pairs examples (situations with correct answers) in supervised learning. The information can be just a simple signal (reward leading to reinforcement) related to the latest interactions. Instead of being given examples of desired behavior the learning agent must discover by trial and error how to behave in order to get the best reward. RL can be viewed as an incremental method for solving a SMDP as discussed in Section 2 without knowing a priory the dynamics of the problem but receiving information over time about the states and the reactions (rewards) to the chosen actions.

A RL system has mainly two components, an agent[3] and its environment:

- The agent (RLAgent) is characterized by:

  knowledge: processed and saved information obtained by communication with the environment that can be used by the agent in order to decide on its behavior.

  learning method: the mechanism by which the agent updates its knowledge.

  behavior: the way the agent chooses to interact with its environment in order to achieve its goal. It is also called the agent's policy, that matches the perceived state from the environment and the taken action by the agent.

  Goal:the agent's criterion, which should be optimized through its behavior.

- The environment is assumed to be in general a Semi-Markov decision process where the actions are controlled by an agent. It is characterized by states, rewards and transitions as introduced in the previous section.

Figure 5 summarizes the communication between the agent and its environment. At each decision moment the agent observes the *current state* (1) of

[3]Although we focus here in a single agent system, multiagent systems in which a group of agents communicate and cooperate had also be considered in the context of RL problems.

the environment and performs an *action* (2) selected according to its decision policy. As a result of the received *action* in the environment (3) a transition to a *new state* takes place and a reinforcement, or *reward* signal is generated (4). The reward signal and the new state are received by the agent (5) and can be used through the agent's learning method in order to update its knowledge about the environment, and consequently it can updates its policy (6). Rewards and state transition functions may be in general stochastic, and the underlying probability distributions are assumed not to be known to the agent.

In this context we can situate our Order Acceptance problem with incomplete information by considering the decision maker as an agent who has to act in an unknown environment where orders arrive and have to be accepted or rejected. Using the models described in Section 2 we simulate the dynamics of the environment. Then through interaction an agent who does not know about such dynamics, should learn the optimal policy by using RL methods. Specifically in our problem we consider incomplete information in the sense that the agent does not know the frequency of order arrivals. Processing time and reward for acceptance become available upon arrival and acceptance of an order. The agent knowledge representation, learning methods, behavioral structures and related parameters are analyzed in the following subsections. For a good introduction on RL see Sutton and Barto 1998 [13].

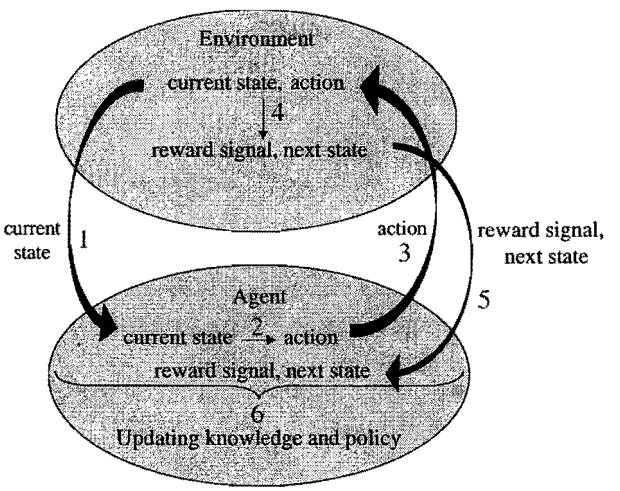

Figure 5: Agent-Environment interaction

3.2 Q-Learning standard methods

Nowadays RL comprises a wide range of problems and related algorithms. As for the problems there are two fundamental categories: the RL evaluation (prediction) and the RL control problems. The first problem concerns the evaluation

of a given policy, that means having the agent following a fix policy to determine certain performance measure of such policy (e.g., the total expected discounted reward). The control problem is a more difficult one and it is to find an optimal policy that maximizes certain performance measure. Our focus here is on the RL control problem, specifically in Q-Learning (QL) methods that look for learning the optimal action value function Q (as defined in Section 2) as a way to obtain an optimal policy. For Semi-Markov processes this is a quite natural approach in our opinion (these are also the most developed and used methods) but we should mention that it is not strictly necessary to do Q-learning in order to solve the RL control problem. Methods that search directly in the policy space (genetic algorithms, genetic programming, simulated annealing) have been also used.

In this subsection we deal with some standard QL methods as found in the literature and we discuss the adaptations made to solve our problem. An important issue is the knowledge representation in these methods. The first classical method is the one-step tabular QL (Watkins 1989) using a backup table representation. In this case each pair state-action (s, a) has an element in the table whose entry is the corresponding approximated action value $Q(s, a)$. However this way of representing knowledge limits the size and complexity of the solvable problems. It is difficult to represent the knowledge (estimated Q-values) for problems with extremely large or even continuous state space. Hence parametrized function approximations may be used that can generalize and interpolate for states and actions never seen before (generalization). An Artificial Neural Network (ANN) is an example of such parametrized function approximations with a massively pararell distributed structure. Such structure and the capability to generalize make it possible for ANN to solve complex problems. See Figure 6 as an example of using artificial neural networks for representing the Q-values. A neural network representing the action-value function Q is not only a function of states and actions but also a function of a parameter vector w, $Q(s, a, w)$. In this case besides the parameters of the learning method, parameters of the neural network must also be tuned.

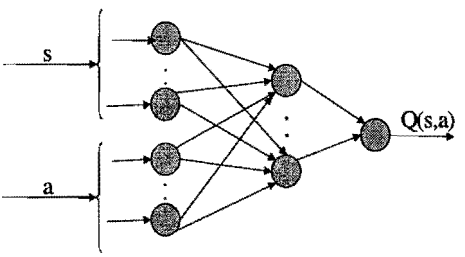

Figure 6: A 2-layer perceptron representation for the Q-values.

The Q-learning was first described by Watkins in his thesis. Several variants of the Q-learning have been proposed since the publication of Watkins thesis in

1989. Here we present some of the most standard tabular methods and their counterpart using neural networks as the parametrized function approximation.

3.2.1 Watkins-QL

The method is based on estimations of the Q-values that are updated after each interaction agent-environment. The agent starts with some estimation (arbitrarily or using some a priory information in case it is available). At each decision moment t in which the environment is in state s_t the agent choose an action a_t based on its current estimation $Q_t(s,a)$ and some specific policy (e.g., greedy policy[4], exploratory policy[5]). The environment reacts to the taken action by giving a reward $r_{t+1} = r(s_t, a_t)$ to the agent and changing to a new state s_{t+1} in a next decision epoch $t+1$ that occurs after $d(s_t, a_t)$ units of time. With this new information the agent updates the Q-values and decides upon a new action a_{t+1} for the present state s_{t+1}, etc. The update rule for this method given the experience tuple $< s_t, a_t, r_{t+1}, s_{t+1} >$ is as follows:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \delta_t \tag{6}$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)$$

where γ is the discount factor, α_t is the learning rate and δ_t is known as the temporal-difference. The idea of this formula is based on the application of the Robbins-Monro stochastic approximation algorithm to estimate an unknown mean (see equation 2) using a single sample, and can be also viewed as a stochastic approximation form of the policy iteration algorithm.

The version of the update rule when using neural networks to approximate the Q-values is based on an update of the weight vector w as follows:

$$w_{t+1} = w_t + \alpha_t \delta_t \nabla_{w_t} Q(s_t, a_t, w_t)$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q(s_{t+1}, a, w_t) - Q(s_t, a_t, w_t)$$

In this case the update rule can be viewed as a gradient descent method in the NN weight space that aims to minimize the temporal-difference term.

Watkins-QL is an off-policy method, meaning that the policy learned about does not need to be the same as the one used to select the actions. In particular it learns about the greedy policy, while the agent follows a policy involving exploratory actions (see section 3.3 for discussion on exploration).

Watkins and Dayan, 1992 [21] showed that in the case of the backup table representation this method converges to the optimum action values with probability 1 as long as all pairs (s,a) are infinitely visited and the learning rate is

[4] A greedy policy is one in which all the actions are greedy. An action a is greedy if $a = \arg\max_{a'} Q(s,a')$

[5] An exploratory policy can choose actions at random which can be useful when there is not enough knowledge, for example at the beginning of the learning process.

reduced over time according to the usual stochastic conditions (see conditions (7) on section 3.3.1). Kearns and Singh (1999 [6]) present the order of sufficient transitions for QL to come within ϵ of the optimal policy in an idealized N-state MDP model where observed transitions are "well-mixed" (same probability of occurrence).

Converge proofs for the case when using neural networks are much more restricted than when using backup table representation. Only for some specific well-structured problems convergence can be guaranteed (see Bertsekas and Tsitsiklis [1] page 337). A problem here is that once the weight vector is updated all the Q-values are also changed and there is no guarantee of reducing the temporal-difference term at each iteration and the algorithm has the potential of divergence.

The application of this method to our prototype problem is quite straightforward since it is a simple model with the number of states depending on the different type of jobs and only two actions that are possible for each state. The agent has always the possibility of choosing one of the two actions depending on its current policy.

In the case of the extended model the size of the action space depends on the number of job types, and not all actions are possible for each state. Here we can think of several variants.

1. The agent receives from the environment together with the information about the state, information about the possible actions. That means that at each decision moment the possibility of allocation in the current capacity profile for each type of job present in the current job list has to be checked.

2. The agent is allowed to choose any of the present jobs. In case the agent chooses for an impossible action for the current state (i.e., the agent chooses a job that does not fix in the current capacity) the agent will be punished with a negative reward. In this case a change has to be made in the environment reward function to consider this punishment term. Also the transition to the next state in case an impossible job is chosen can be made to a new state exactly as the previous one but the impossible job type in the job list. Since the agent is looking for actions that maximize its total (discounted) reward it is reasonable to expect the agent learns avoiding such impossible actions that give negative rewards.

In our approach we choose for the second variant since it is numerically somewhat more efficient.

Also for the extended model the following Sarsa variant of the update rule is interesting since avoids the evaluation of the Q function over the action space.

3.2.2 Sarsa-QL

Sarsa QL is an alternative for the Watkins update rule. This method was introduced by Rummery and Niranjan, 1994. The name SARSA refers to the

use of $< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} >$ for the update rule. The update rule can be viewed as an action-sampled version of the Watkins update rule (6). Unlike Watkins-QL, sarsa is an on-policy method meaning that the policy learned about through the updating rule is the same that is used to select the actions. This SARSA update rule is cheaper to compute than Watkins rule since it does not involve a $\max_a Q_t(s_{t+1}, a)$ computation which can be convenient in problems with a large action space. But on the other hand it is more sensitive to exploration strategies than the off-policy methods since it learns from the same policy that produces the actions. The update rule for this method given the experience tuple $< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} >$ is as follows:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \delta_t$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t, a_t)} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

The convergence of SARSA was proved by Singh, Jaakkola, Littman and Szepesvari, (1997 [16]) for the backup table case. The neural network version for the update rule is just as for the Watkins case but now changing the temporal difference term with the sarsa idea:

$$w_{t+1} = w_t + \alpha_t \delta_t \nabla_{w_t} Q(s_t, a_t, w_t)$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t, a_t)} Q_t(s_{t+1}, a_{t+1}, w_t) - Q_t(s_t, a_t, w_t)$$

Parameters of the learning method and on NN representation are discussed further in subsection 3.3.

3.3 RL Parameters

A RL approach involves also an interesting parameter setting dimension. A lot of tuning could be necessary before one finds an efficient parameter setting. Here we consider three different parameters. These parameters concern learning, exploration and the knowledge representation structure used for updating. We discuss these issues in the next two paragraphs.

3.3.1 Learning and exploration parameters

Learning and exploration rates are key issues in reinforcement learning.

The learning rate or step-size α_t is a measure of how fast new information is incorporated in the general knowledge. In the previous section $\alpha_t \in [0, 1]$ was introduced, but its behavior over time with was not discussed. There are well known conditions for the learning rate from the general stochastic approximation theory that are also used in the convergence proof of the QL methods:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty \tag{7}$$

The first condition is to guarantee that steps are large enough to overcome initial conditions or random fluctuations and the second is to guarantee that steps become small enough to assure convergence. Sequences of step-size parameter that meet both conditions often converge very slowly or need a lot of tuning in order to obtain satisfactory convergence rates. It is surprising how many successful applications usually do not use step-sizes satisfying these conditions but for example constant step-size. Using a constant learning rate the second conditions is not meet, and the learned function will never completely converge but will vary in response to the most recently received rewards. However, this can be desirable in case of nonstationary environment. The conditions also require separate learning rate for each state-action pair which again could be no very practical for big size problems. In our experiments we use decreasing time-varying learning parameter independent of the state-action pair.

Exploration increases experience by for example choosing actions at random. Exploitation deals with the use of the available knowledge by for example choosing the greedy actions. The known exploitation-exploration trade-off (see Thrun 1992 [19] for a survey on exploration) can be briefly described as follows. If we always exploit the actual knowledge taking the optimal action with respect to the actual Q-values without exploration, many relevant state-action pairs may not be visited. That is why mainly at the beginning when the knowledge we have is not very accurate yet, we need to explore as much as possible.

Efficient exploration is fundamental for learning. Too much exploration can cause nearly random behavior and too less can lead to non-optimal solutions. Some possible exploration strategies are shown below:

interval-based exploration: Switches between an exploration interval (choose actions at random) and next greedy actions are employed.

$\varepsilon-$greedy exploration: With probability 1-ε one chooses a greedy action and with probability ε chooses a random action.

Boltzmann exploration: In state s, and action a is chosen with probability

$$\frac{e^{-\frac{Q(s,a)}{T}}}{\sum_{a'} e^{-\frac{Q(s,a')}{T}}}$$

where T is called "*temperature*" parameter and T should be decreased during the course of the algorithm.

For reasons of simplicity we use in our experiments the $\varepsilon-$greedy exploration rule with an ε parameter decreasing over time which is also the most widely used strategy in the literature.

3.3.2 Bootstrapping degree: The λ parameter

Here we consider a class of more advanced updating rules. Instead of updating the estimated value function based solely on the approximated value of the

immediate successor state (see for example update rule 6), one can think of methods that base the updates on an exponential weighting of values of future states as suggested by the definition of $V^{\pi}(s)$ in section 2.1. This is precisely the idea of methods that introduce the λ parameter as the weighting factor. It is like looking back in the time and correcting previous predictions by using the information on actual states. It is done by means of the eligibility trace that can be viewed as a temporary record of the occurrence of an event that make it eligible for further learning. Given the structure of an optimal policy eligibility can only occurs, if the greedy action is taken.

Eligibility traces provide a link between Monte Carlo ($\lambda = 1$) and one-step RL methods ($\lambda = 0$).

In this way the parameter λ trades off between bias and variance. The target values used when $\lambda = 1$ are unbiased samples but may have significant variance since each depends on a long sequence of rewards from stochastic transitions. By contrast when $\lambda = 0$ the target values have low variance since the only random component is the reward of a single transition, but are biased by the inaccuracy of the current estimated values.

It is the belief that the λ parameter (for $\lambda \neq 0$) as distributing credit throughout sequences of actions, leads to faster learning. Experimental results (Sutton1988) for a specific prediction problem show that intermediate values of λ that are closer to 0 give the best results. However a full understanding of how λ influences the rate of convergence is yet to be found. Van Roy [20] has suggested that it could be desirable to tune λ as the algorithm progresses but even both directions (starting with $\lambda = 0$ and approaching 1, and the other way around) have been advocated. In our experiments for the prototype type problem we shall take in some cases $\lambda > 0$, λ constant.

For the case of a neural network representation of the Q-values eligibility traces do not correspond to each state-action pair but a trace is defined for each component of the parameter vector. In this case it can be interpreted as a smooth parameter change proportional to the gradient function. Next we show the update rules for the previous methods when considering this λ parameter.

Watkins-Q(λ):

$$
\begin{aligned}
Q_{t+1}(s,a) &= Q_t(s,a) + \alpha_t \delta_t e_t(s,a) \\
\delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t) \\
e_t(s,a) &= I_{s,s_t}\ I_{a,a_t} + \begin{cases} \gamma^{d(s_t,a_t)} \lambda e_{t-1}(s,a) & \text{, if } a_{t-1} \text{ was greedy} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

Sarsa (λ)

$$
\begin{aligned}
Q_{t+1}(s,a) &= Q_t(s,a) + \alpha_t \delta_t e_t(s,a) \\
\delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t) \\
e_t(s,a) &= I_{s,s_t}\ I_a, a_t + \gamma^{d(s_t,a_t)} \lambda e_{t-1}(s,a)
\end{aligned}
$$

Watkins-Q(λ, w)

$$w_{t+1} = w_t + \alpha_t \delta_t z_t$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t,a_t)} \max_a (Q(s_{t+1}, a, w_t)) - Q(s_t, a_t, w_t)$$
$$z_t = \nabla_w Q(s_t, a_t, w_t) + \begin{cases} \gamma^{d(s_t,a_t)} \lambda z_{t-1} & \text{, if } a_t \text{ was greedy} \\ 0, & \text{otherwise} \end{cases}$$

Sarsa-Q(λ, w)

$$w_{t+1} = w_t + \alpha_t \delta_t z_t$$
$$\delta_t = r_{t+1} + \gamma^{d(s_t,a_t)} Q_t(s_{t+1}, a_{t+1}, w_t) - Q_t(s_t, a_t, w_t)$$
$$z_t = \nabla_w Q(s_t, a_t, w_t) + \gamma^{d(s_t,a_t)} \lambda z_{t-1}$$

$I_{x,y}$ is a comparison function[6]. In the extended case of section 2.2 we work with a slight generalization for λ which follows logically from the theory in section 2:

$$\lambda(s,a) = \begin{cases} 1 & a \neq 0, rew(s,a) > 0 \\ 0 & a \neq 0, rew(s,a) < 0 \\ \varrho & a = 0 \end{cases}$$

where $\varrho \in (0,1)$. The first case is when a job is accepted that can be allocated, the second case is when a job is accepted that cannot be allocated and the third case is when the job list is rejected.

3.3.3 Neural network architecture parameters

As discussed before neural networks are a rather general parametrized function approximation to represent the Q-values. The most widely used neural networks are multilayer perceptrons type. Multilayer perceptrons are global in the sense that they compute their output using all their parameters. Training methods work by adjusting all the parameters with each training instance. In the application of multilayer perceptron matters like codification of the inputs and number of hidden nodes are to be defined that can influence the success of the application. For reasons of simplicity we will focus here on multilayer perceptron with one hidden layer.

The inputs of the NN are the state of the system s and an action a for that state. The output is then the corresponding Q-value $Q(s,a)$ (see Figure 6).

For the prototype model the state of the system has to be defined by some information of the arriving job type(processing time, reward for acceptance, or just an index identifying the job type). For the action a single binary input was used that is one for acceptance and zero for rejection. For the input codification we tried several variants. In the case of N job types we considered:

[6] $I_{x,y} = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise} \end{cases}$

1. Each state is represented by N binary inputs, one per job type. In this case we are using a priory information about the number of different type of jobs, and the given information is just like the job type index as in the backup table case. The input size in this case scale with the size of the problem.

2. Each state is represented by 2 real value inputs, one for the processing time and one for the reward of the job type. This representation is independent of the problem size.

In our approach we choose to work with codification 1 for several reasons. Codification 2 really allows for an ∞-dimensional generality and is too general in that respect. Moreover, codification 1 generalizes easily to the extended case from Section 2.2. Using m hidden nodes, and N type of jobs the neural network contains $(N+3)m+1$ parameters. Hence representation one would always use more parameters than values to be learned $(2N)$ if $m > 1$. Therefore for the prototype problem it makes only sense to use the backup table approach or to use $m = 1$ as a lower dimensional representation. Nevertheless, a neural network with $m \geq 2$ (hence with more unknown parameters) should no harm and we did experiments also of this type to confirm this.

In the case of the extended model the state is defined by the current job list and the current capacity profile. For the actions in this case we can think of an integer input that takes values from 0 to N indicating rejection of the current job list or the type of job chosen. For the states we have the following generalization of the prototype:

N integer inputs showing the amount of jobs of each type.

H integer inputs showing the already allocated capacity at each of the capacity profile stages.

A 2-layer perceptron with m hidden neurons has (N+H+3)m+1 parameters (weights) which is in this case a much lower dimension than the state and action space in general ($|SxA| = (N+1)(Cmax+1)^H \prod_{i=1}^{N}(m_i+1) = O(NM^{H+N})$ $M \geq 2$).

3.4 Speeding up RL by Model-based learning

An important issue in learning techniques is the learning speed. Besides the bootstrapping idea there are several other QL methods that aim to speed up the standard QL. Here we focus at a model-based approach.

So far we have discussed model-free RL methods where an agent learns about an optimal policy without never knowing explicitly the model of the environment it interacts with. However these methods may have been using inefficiently the data they gather and therefore needing too much time to achieve good performance. Model-based methods are alternative methods in which the agent incorporates in its learning process information he gathers about the model.

There is an open debate about which methods are better, model-based or model-free methods. A real fact is that although model-free methods use fewer computation time per experience they do not exploit all the information they gather and they make inefficient use of it. This could be one of the reasons for the large amount of experience these methods need in order to obtain a good performance. In this sense it is the believe that incorporating some knowledge from the accumulated experience in the learning procedure can in fact help with the learning speed. But model-free methods are much simpler and are not affected by biases in the design of the model. Recently Kerns and Singh [6] argued that both methods have roughly the same sample complexity (rather rapid convergence to the optimal policy as a function of the number of states transitions observed).

Model-based methods unlike model-free need to store besides the value function, the domain model. This is usually done by means of backup tables which scale with state and action spaces. Dynamic Bayesian networks can be used in many cases to represent models in compact form (see [17]). In our OA models with incomplete information we assume that the only model parameters to be learned are the arrival frequencies of each job type, then a single table structure is sufficient in this case.

There are also different versions of this model-based approach. In our approach the idea is not to make a separation between the learning phase and the solution phase but taking a closer idea to RL methods. At each time step the model parameters ($\widehat{rew}(s,a)$ and $\widehat{\Pr}(s,a,s')$) are estimated and used in the update rule with:

$$\delta_t = \widehat{rew}(s_t,a_t) + \gamma^{d(s_t,a_t)} \sum_{s'} \widehat{\Pr}(s,a,s') \max_a Q(s',a) - Q_t(s_t,a_t) \qquad (8)$$

In the case of the prototype model this approach is not very difficult due to the simplicity of the transition probabilities. However for the extended model even if we know the arrival frequency per job type it is not a straightforward matter to compute the summation in δ_t since it contains too many terms. For this reason another alternative will be applied to which we refer as Dyna. Therefore we briefly present the Dyna method which is a very intuitive and simple approach.

Dyna-Q (Sutton 1990) is an architecture that combines model learning and direct QL. It simultaneously uses experience to estimate a model and uses the experience and the estimated model to learn the optimal policy. The method uses one of the update rules as previously defined. However, after the first updates (model and value function) with the data from the interaction between agent and environment, the agent includes additional updates of the estimated value function by simulating data from the estimated model. In our case this means by simulating a prescribed number of arrivals patterns besides the actual one occurring in s_{t+1}.

This approach can easily be used for both our models. For the case of the extended model we do not need the complete model but only a sample model.

The arrival frequencies can be estimated and together with the procedures that determine the state transitions, they can be used to obtain simulated data.

4 Experimental results

In this section we present the application of Reinforcement Learning to the order acceptance problems described in Section 2. Here we use QL methods that aim to approximate the optimal Q-value function as discussed in Section 3.

First in section 4.1 we analyze a simple case of the prototype problem as presented in Section 2.1 just to get some insight into the application of the algorithms. Even for this simple problem extensive experimentation was necessary in order to find appropriate parameters. In section 4.2 we present experimental results with bigger size cases of the prototype problem. In Section 4.3 we show some first results on experiments for the extended model as presented in section 2.2

Conclusions from the experimental study are presented in section 5.

4.1 On the prototype problem: small case.

We consider a simple instance with 3 job types. The table below shows the problem definition:

job type	probability of arrival at the decision epoch	processing time	reward
1	0.4	2	3
2	0.5	4	2
3	0.1	5	8

According to the prototype model in Section 2.1 the states of the model are defined by the type of job arriving when the server is idle. At each decision moment the possible actions are accept or reject the arriving job. The goal is to find a policy that maximices the total expected discounted reward (the discount factor is 0.9). Using the complete model information the optimal policy can easily be found by policy iteration. The table below shows the optimal policy and the corresponding Q-values:

state	**Q*(state,reject)**	**Q*(state,accept)**	**optimal action**
1	10.7816065791759	12.7034459212583	accept
2	10.7816065791759	9.85979119621925	reject
3	10.7816065791759	15.0738120765973	accept

In this case the optimal policy π^* rejects an arriving order only if it is of type 2.

In each experiment we compute averaged performance criteria over 10 simulation runs and compare this performance to the optimal solution. We use the following performance criteria:

- MSE-Q_t: The mean square error of the (learned) actual Q_t-values compared to the (optimal) Q^*-values at time t.
- MSE-Q^{π_t}: The mean square error of the Q^{π_t}-values corresponding to the (learned) policy π_t at time t, compared with the (optimal) Q^*-values. The learned policy π_t is defined as the greedy policy with respect to the learned Q_t-values. Note that this measure provides implicit information on the probability that the optimal policy, or a policy close to optimal is found after t iterations in the learning process.
- AvToRew$_t$: The total average reward accumulated per unit of time upto time t.

A suboptimal policy π' that accepts all the jobs has a mean square error in the Q-values of 1.63 compared to the optimal Q-values (MSE-Q^π). The table below shows this suboptimal solution.

state	**suboptimal policy** (action)	**Q**(state,reject)	**Q**(state,accept)
1	1	9.3458	11.4112
2	1	9.3458	8.8131
3	1	9.3458	14.1318

Notice that in the case the Q-values for policy π' would have being learned at time t, looking at the greedy policy with respect to these values ($\pi_t(s) = \arg\max_a Q^{\pi'}(s,a)$) one obtains the optimal policy $\pi_t = \pi^*$ which rejects only orders of type 2. This illustrate a situation where MSE-Q_t ($= 1.63$) differs from MSE-Q^{π_t} ($= 0$).

Figure 7 shows the total average reward per unit of time of two different agents that follow these policies. We use these values to compare with the results of agents that use QL methods.

In the next Sections we present the application of QL methods to this problem for the case of incomplete information. First as a preparation the naive model-based approach and the backup-table method are discussed, next we consider neural networks knowledge representation methods.

4.1.1 Q-L experimental results with backup tables

In this Section we present experiments to analyze the influence of the parameter settings for different methods in our problem. A first study shows the results of using different learning and exploration parameters with the bootstrapping parameter set to zero. In a second study we use fixed learning and exploration parameters corresponding with good performance from the first study, in order to analyze the influence of the bootstrapping parameter. As we mentioned before the most straightforward idea for a learning agent in this problem is the naive model based approach (or two phase).

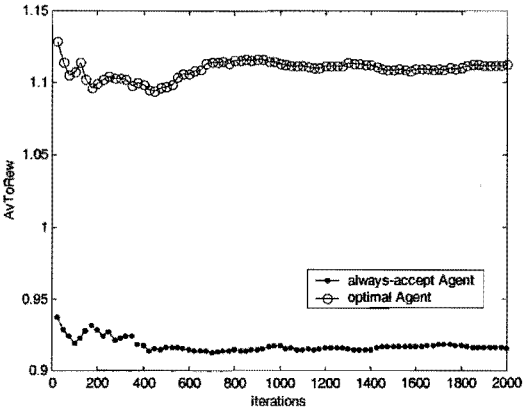

Figure 7: Total average reward per time of two agents following fixed policies, the *optimal* policy and the *"always accept"* policy

Naive model based approach: Here at each iteration (job arrival+decision making+reward for decision) the agent updates the frequency of arrival and reward value of the arriving job and the decision. The estimated model is solved using policy iteration. Due to the structure of the model this learning process is independent of the followed policy.

Results in Figure 8 show that 50 iterations were sufficient to learn the optimal policy with a MSE-Q$\approx$ 1.

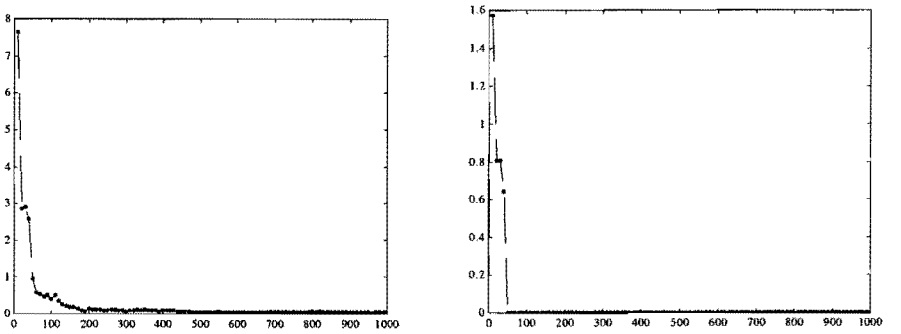

Figure 8: Naive model-based in the 3 jobs problems. The left graph shows the MSE-Q and the right one the MSE-Q^{π}

Study 1: influence of learning and exploration parameters. Here we present the results from experiments with some representative combinations of learning and exploration parameters. We refer to α_k, ε_k as the learning and exploration parameter at moment k of the learning process. Results from the naive model based indicate that $+/-50$ iterations are necessary in order to estimate the arrival rates accurately enough that a good estimate of the Q-values leads to the optimal policy. For our Q-learning method this suggests strongly that we should keep a certain level of exploration till such sufficiently accurate estimates are possible.

We use curve decreasing functions for both parameters as follows

$$\varepsilon_k = \frac{\varepsilon_0}{1+\frac{k}{T_e}}, \quad \alpha_k = \frac{\alpha_0}{1+\frac{k}{T}}$$

where ε_0 and α_0 are initial values and T_e, T define the decreasing speed. Note that the parameters drop to half its original value when $k = T, T_e$ respectively.

In the experiments we explore how initial and speeding parameters should be set. Below we show some results for $T_e = 100$ and $\alpha_0 = 1, 0.5$ and $T = 200, 100, 50$, respectively. Besides we add another value for T_e ($T_e = 200$) in order to asses our choice for $T_e = 100$. Summarizing, we use the following 7 parameter combinations:

1	$T_e = 100$	$\alpha_0 = 0.5, T = 200$
2	$T_e = 100$	$\alpha_0 = 0.5, T = 100$
3	$T_e = 100$	$\alpha_0 = 0.5, T = 50$
4	$T_e = 100$	$\alpha_0 = 1, T = 200$
5	$T_e = 100$	$\alpha_0 = 1, T = 100$
6	$T_e = 100$	$\alpha_0 = 1, T = 50$
7	$T_e = 200$	$\alpha_0 = 1, T = 100$

In our first experiments we use the standard Q-learning method introduced by Watkins (Twatkins stands for Table-Watkins).

Twatkins Figures 9, 10 and 11 show results from the performance measures MSE-Q, MSE-Q^π and AvToRew. Each graph is an average over 10 different simulation runs with 2000 iterations each.

Figure 9 represents the MSE-Q curves for the first six combinations. Combination 2 with $T = 100$ leads to the best MSE-Q performance for $\alpha_0 = 0.5$. In case $\alpha_0 = 1$ we find even better results, again for $T = 100$.

The results from the MSE-Q^π performance in Figure 10 match pretty well with the results of the MSE-Q performance. It shows that each combination shows rather stable behavior after learning the optimal policy.

The results from the Average total reward in Figure 11 show the advantage of using no more exploration than necessary.

To conclude this survey we mention that a similar study was made with Sarsa(0), instead of Twatkins, but no significant differences were noticed, just a deterioration in the learned Q-values (the MSE-Q was always above 1). This behaviour may be due to the sensitivity of Sarsa methods for the followed policy.

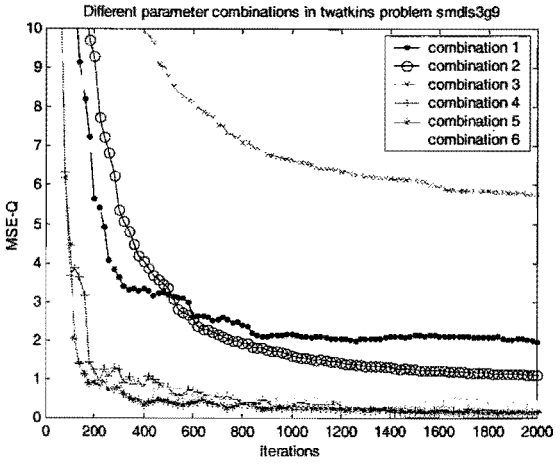

Figure 9: MSE-Q performance. All combinations use same exploration parameter ($T_e = 100$). The first 3 use $\alpha_0 = 0.5$ and $T = 200, 100, 50$. The last 3 use $\alpha_0 = 1$ and $T = 200, 100, 50$

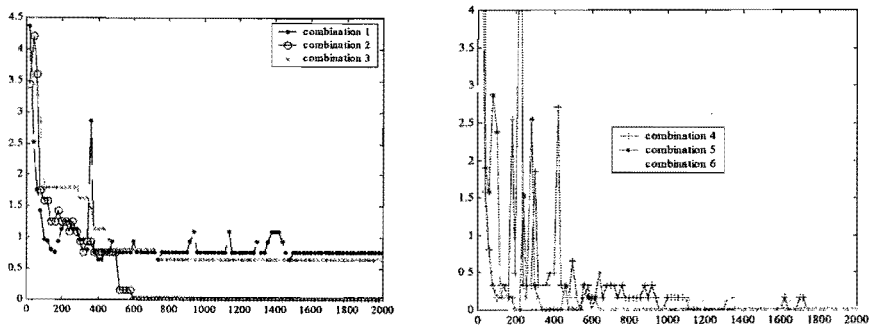

Figure 10: MSE-Q^π performance in a 3 jobs problem. All combinations use same exploration parameter ($T_e = 100$). The first 3 (in the left) use $\alpha_0 = 0.5$ and $T = 200, 100, 50$. The last 3 (in the right) use $\alpha_0 = 1$ and $T = 200, 100, 50$

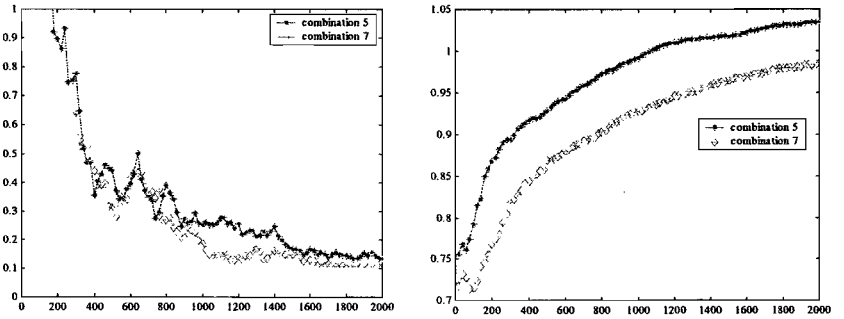

Figure 11: Both combinations use same learning parameter ($\alpha_0 = 1$,$T = 100$) but different exploration parameter ($T_e = 100, 200$). In the left the MSE-Q and in the right the AvToRew.

Study 2: influence of the lambda parameter Using the parameter combination $T = T_e = 100$, $\alpha_0 = 1$ from the first study, we tested the influence in the learning procedure of different values of the lambda parameter $(0, 0.3, 0.5, 0.7, 0.9, 1)$. For no one of the methods introducing $\lambda > 0$ helps to improve the final values learned for the Q-values. Bootstrapping should speed up the learning process and this effect is found indeed early in the learning process. Figure 12 demonstrates that learning reasonable Q-values can be done faster for example with $\lambda = 0.6$ compared with $\lambda = 0$ in the first 100 iterations. Therefore bootstrapping could be considered in more complex problems or in case of NN in order to speed up the learning process.

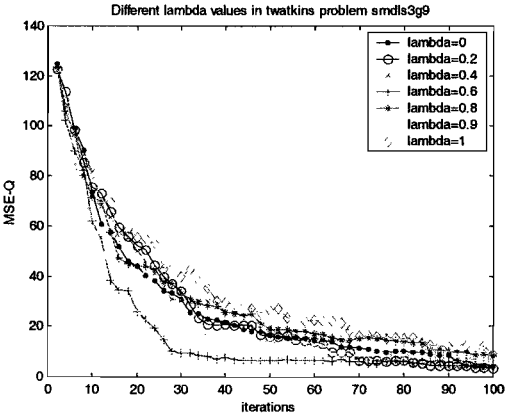

Figure 12: MSE-Q performance in the first 100 iterations using different values for λ.

4.1.2 Q-L experimental results with neural networks

In this Section we present some results for the prototype problem with 3 job types using a neural network approach as explained in section 3.2. As in section 4.1.1 values have to be set for the learning, exploration and bootstrapping parameter. Whereas previously $\alpha_0 = 1$ was a good choice, it turns out that now working with $\alpha_0 = 0.1$ ($T = T_e = 100$) is a better choice. This small value for α_0 compensates for the fact that now by changing the parameters w in the NN all Q-values are changed and not just the most appropriate ones as in the backup table method. Some tests on parameters settings were necessary to find out that these settings give an acceptable solution, but we shall not report on that here. Instead we shall focus mainly on the effect of the NN structure. Figures 13, 14 and 15 show the three performance measures previously defined, (MSE-Q, MSE-Q^π and AvToRew) for 5 RLAgents that use different sizes of the NN for the knowledge representation ($1, 2, 3, 4, 5$ hidden neurons i.e. NN architectures with $7, 13, 19, 25, 31$ parameters, respectively). In this section we restrict ourselves to the Watkins updating rule, since as explained in section 3.2, SARSA could only be advantageous in case of a large action space.

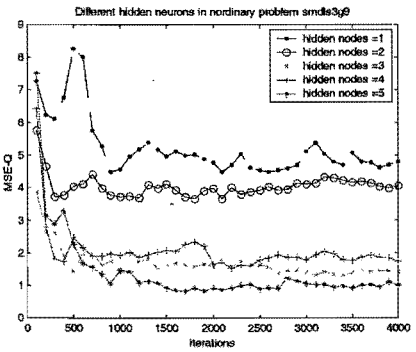

Figure 13: MSE-Q performance for different sizes of the NN .

It is clear, that the quality of the results increases with the number of hidden neurons. We want to emphasize that this is rather unexpected, since as discussed in section 3.3, already for 2 hidden neurons the number of parameters in the NN exceeds the number of Q-values in this simple case. A "decreasing rate of return " effect for these excess parameters would be reasonable in the long run. It is an interesting observation that the NN is capable to exploit the excess degrees in freedom for finding a more efficient route to good approximations. Notice that except for the case of only one hidden neuron, the rest of the cases show an average total reward superior to the "always accept" policy.

Figure 16 shows results with different λ values. In these cases the same λ value was used through each run. Again there was no significance difference between different λ values in the quality of the final result for the Q-values, but

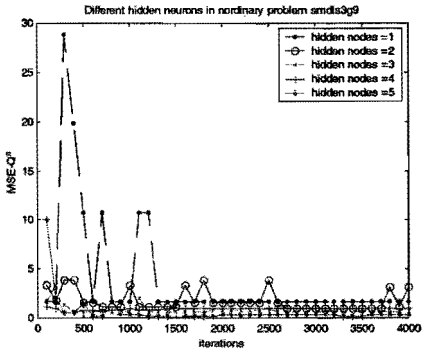

Figure 14: MSE-Q^{π} performance for different sizes of the NN.

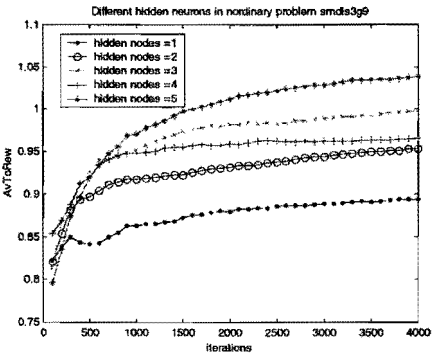

Figure 15: AvToRew performance for different sizes of the NN.

bootstrapping values in the range 0.3 to 0.5 speeds up the learning process.

4.1.3 Final Comparison

Figure 17 shows a final comparison between the total average reward for different agents. All the learning agents outperformed the greedy policy and have a tendency to approximate the optimal one. Dyna and Naive that incorporate model base ideas have a faster learning than the direct RL.

As discussed in previous Section Dyna is a RL method that combines direct (model-based) and indirect RL. For this simple problem the estimated model can be store in tables. We have assumed that the general structure of the model is known (it means that it is known that the environment follows a SMDP as the prototype model). In this case the only thing to be estimated from the model is the frequency of arrival per job type.

At each iteration in the standard Q-learning method, the Dyna method

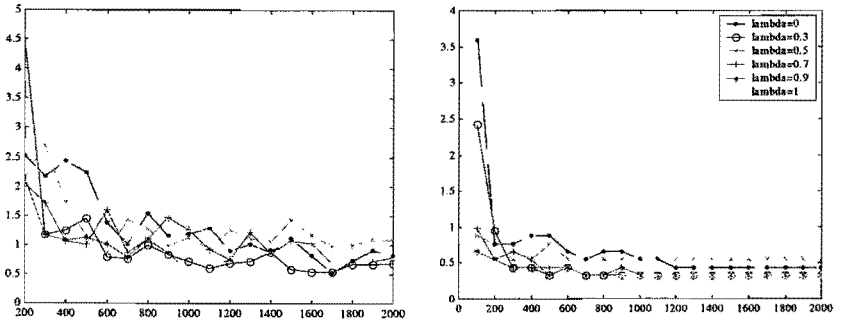

Figure 16: Nwatkins in the 3 jobs problems with different λ. The left graph show the MSE-Q and the right one the MSE-Q^{π}

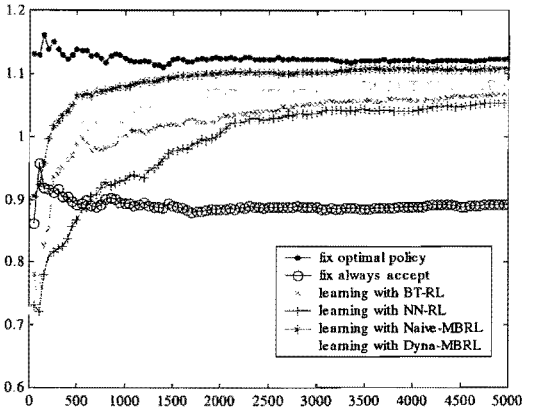

Figure 17: Agents comparison in a 3-jobs problem.

includes M extra simulated iterations by taking 'experience' from the estimated model. The amount M of simulated experiences could be depending on the iteration number t. In our experiments we use $M = max(100, t)$ at time t.

4.2 On the prototype problem: other cases.

In this Section we present experimental results in problems with 5 and 10 jobs. The parameters for both problems were:

$\alpha_k = 5/(500 + k), \varepsilon_k = 200/(200 + k)$

Figure 18 shows the comparison between agents using different neural network architecture. The difference is in the number of hidden neurons. The results show that using 1 hidden neuron give no good results but from 2 hidden neurons on results get better. For the case with 1 hidden neuron the NN loss the generalization capabilities and is not very flexible. Overdimension in this problem by using a 2-layer perceptron with more that a hidden unit is unavoidable. $|SxA| = 2N$ is a polynomial grade 1 with respect to N (the number of jobs) as it is $(N+3)m+1$ the number of parameters (weights) in the NN, where m is the number of hidden neurons. And $m(N+3)+1 > 2N$ when $m > 2$.

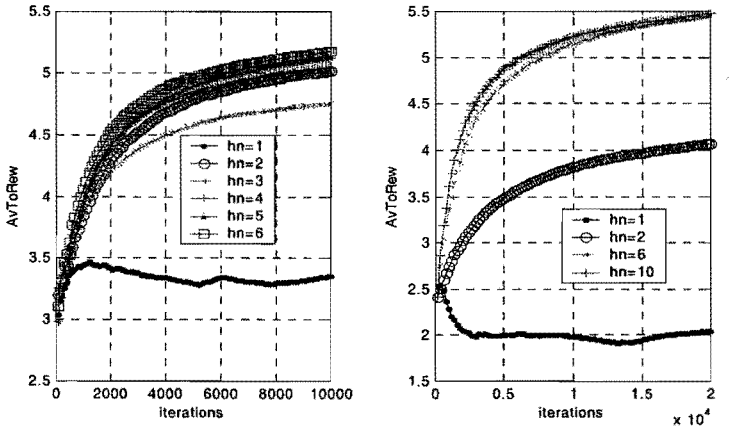

Figure 18: Different RL-Agents in problems with 5 (left graphic) and 10 (right graphic) jobs.

Figure 19 shows the comparison of 4 learning agents (RL-NN,RL-BT, Naive-MB, Dyna-MB) with 2 agents using fix policies ('always accept ', optimal). As in the experiments with 3 jobs all the learning methods outperformed the greedy policy 'always accept' and have a tendency to approximate the optimal one. Again as expected the model base ideas speed up the learning process.

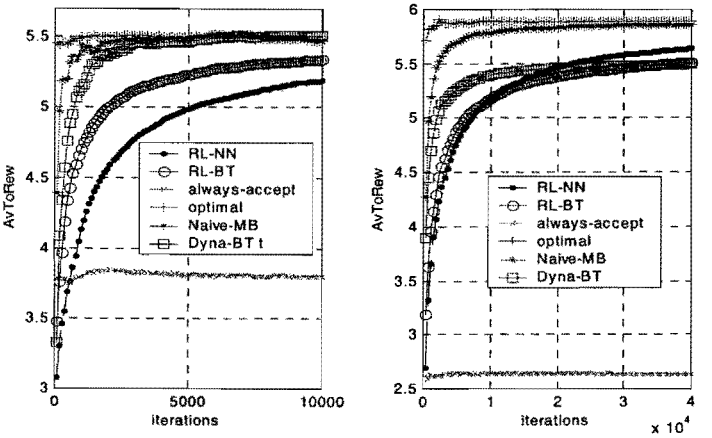

Figure 19: Agent comparison in problems with 5 (left graphic) and 10 (right graphic) jobs.

4.3 On the extended problem: small case

In previous Sections we introduce the extended order acceptance problem, in Section 2.2 we give the model definition and in Section 3 we give comments about the application of Q-learning to this problem. In this Section we show some preliminaries experimental results with a small case.

In this model the cardinal of the state-action space is: $|SxA| = (N + 1)(Cmax + 1)^H \prod_{i=1}^{N}(m_i + 1) = O(NM^{H+N})$ $M \geq 2$ which indicates the complexity of using traditional methods from Dynamic Programming or the use of RL with backup tables. Here we experiment with RL using NN and compare the results with some heuristic policies.

Problem 1 Let us consider a problem instance as follow:

job type i	t_i	w_i	r_i	m_i	
1	1	2	20	2	$C_{max} = 2$
2	3	5	5	2	$H = 3$

where t_i,w_i,r_i are (as defined in section 2.2) the due date, work request and reward of job type i. m_i is the maximum number of arrivals of job type i to be consider in the unit of time, more jobs over this amount are neglected.

This problem has 243 states ($(2+1)^3 * 3^2 = 243$) and the action space has size 3. A negative punishment reward can be defined as -20 when the agent choose to accept a job that does not fix in the current capacity profile. The perturbation distribution is defined as follow:

Perturbation value	Prob.	Penalty
0	$\frac{1}{3}$	0
1	$\frac{1}{3}$	$\begin{cases} -1 & c_1 = 2 \\ 0 & otw \end{cases}$
-1	$\frac{1}{3}$	0

As we have argued before (see Section 3) we use Sarsa method for this model. Parameters were chosen from the previous experience on the prototype model. $\alpha_k = 5/(500+k), \varepsilon_k = 100/(100+k)$ and 20 hidden neurons. In this case the approximation function (NN) is not overdimensionated, the NN has $(H+N+3)*m+1 = (3+2+3)*20+1 = 161$ parameters in order to learn the 729 values of the state-action value function Q(s,a).

The total average reward of the RL agent was compared with the total average reward of agents that follow some heuristic policies. In the total average reward we should not count the punishments since they are only meant to teach the agent to avoid impossible actions (to accept jobs that do not fix in the capacity profile). Figure 20 shows a comparison of the RL agent with two heuristics:

- Greedy reward/work: choose at each decision moment from the present jobs the job with highest reward/requested_capacity.
- Elitist: always choose job type 1 if it is present, otherwise reject everything else. This heuristic is elitist in the sense that only choose the job with highest reward/requested_capacity. This rule needs to know this ratio for all the job types in advance.

By an analysis of the dynamic of the system the state space was reduced from 243 states to 28 states. The number of values Q(s,a) to be found is reduced to 11 by using the Bellman equation. In the reduced system is easy to prove that the elitist policy is optimal for this problem.

In Figure 20 the RL-agent outperforms the greedy agent and shows a tendency towards the elitist agent which follow an optimal policy for this specific problem.

5 Conclusions

As a simple example, the prototype model helped us as an introduction for the application of the RL approach. In the experiments backup tables were less sensitive to parameters settings than the neural networks. Including model based ideas is also a trivial way for speeding up the learning process but it increases the computational time per iteration steps. No relevant benefits were found in the final performances by using a bootstrapping parameter different than zero and constant during the learning process. Only a faster improvement of the learned Q-values was noticed for some λ values in the early stages of the learning process. A variable λ, say decreasing with the iteration number is

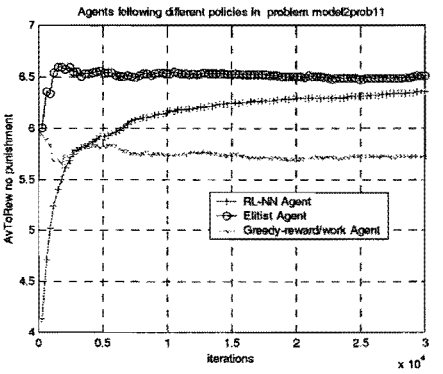

Figure 20: Comparison of 3 agents for a 3 jobs problem in the extended model

an alternative to try in future experiments. Watkins update rule gave slightly better results than the SARSA for the prototype problem. In all the cases the RL strategies outperformed the simple heuristic rule *always accept*. RL approach seems to be suitable for OA problems dealing with opportunity cost and uncertainty

We expect for the extended model to find intelligent rules in the policy obtained with QL methods. We believe the RL approach is a very flexible method that could help in the search for good order acceptance rules.

Further studies could analyze extended models and the possibility of including OA-RL strategies in an integrated capacity planning approach.

References

[1] Bertsekas, D. P. and J. N. Tsitsiklis. Neurodynamic Programming. Athena Scientific, 1996.

[2] de Boer, Ronald. Resource-Constrained Multiproject Management. A hierarchical decision support system. Ph.D. Thesis. University of Twente. 1998.

[3] Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, M. E. H., editor, Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pages 1017–1023, Cambridge, MA. MIT Press.

[4] Garbe, R., Algorithmics aspects of interval orders, PhD. Thesis, University of Twente, Enschede, 1996.

[5] ten Kate, H.A., Order acceptance and production control. PhD. Thesis, Rijksuniversity Groningen, 1995.

[6] Kearns, M., S. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In Neural Information Processing Systems 12, 1998. MITT Press 1999.

[7] Nawijn, W. M. . The Optimal Look-Ahead Policy for Admission to a Single Server System.Operations Research Vol. 33, No. 3, May-June 1985.

[8] Kaelbling, L.P., Littman, M.L., and Moore, A.W., Reinforcement Learning: A survey, Journal of artificial intelligence research, 4, 237-285, 1996.

[9] Peng, J. and R. J. Williams. Incremental Multistep Q-learning. Machine Learning, 22 (1996), pp, 283-290.

[10] Pinedo, M.,Chao X., Operations Scheduling with applications in manufacturing and services. McGraw Hill 1999.

[11] Puterman, M. L. . Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, 1994.

[12] Raaymakers, W., Order acceptance and capacity loading in batch process industries. Pd.D Thesis. Technische Universiteit Eindhoven. 1999.

[13] Sutton, R.S., and Barto, A.G., Reinforcement Learning: An introduction. MIT Press, Cambridge, Massachusetts, 1998.

[14] Sutton, R. Open Theoretical Questions in Reinforcement Learning. 1999.

[15] Singh S, Bertsekas D. (1997), Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems . Appears in NIPS 10 proceedings.

[16] Singh, S. P., Jaakkola, T., M.Littman and C.Szepesvari. On the convergence of single-step on-policy reinforcement learning algorithms. Machine Learning, 1997.

[17] Tadepalli, P., KoKyeong OK. Model-based Average Reward Reinforcement Learning. 1998

[18] Tesauro, G. J. . TD–gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation, 6(2):215–219. 1994

[19] Thrun, S. (1992). The Role of Exploration in Learning Control. In Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches, Van Nostrand Reinhold, Florence, Kentucky 41022.

[20] Van Roy, B. Learning and value function approximation in complex decision processes. Ph.D. Thesis 1998

[21] Watkins, C. J. C. H., Dayan, P. Q-learning. Machine learning, 8: 279:292. 1992.

[22] Wester F. A.W., J.Wijngaard and W.H.M. Zijm. Order acceptance strategies in a production to orderenvironment with setup times and due dates. International journal of production research, vol.30, pp.1313-1326. 1992.

[23] Wiering, M. and Jurgen Schmidhuber. Speeding up Q(λ)-learning. ECML 98.

[24] Wiering, M.,Explorations in Efficient Reinforcement Learning. Ph.D. Thesis. Amsterdam University, 1999.

[25] Winston, W. L. Operations research: Applications and algorithms. Duxbury Press. 1994.

[26] Wouters, M.J.F., Relevant cost information for order acceptance decisions. Production Planning and control, 1997, vol. 8, No. 1, 2-9

[27] Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pages 1114–1120.